PTO-1390 (Rev. 9-2001) U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE		ATTORNEY'S DOCKET NUMBER 1178.P002US
TRANSMITTAL LETTER TO THE UNITED STATES DESIGNATED/ELECTED OFFICE (DO/EO/US) CONCERNING A FILING UNDER 35 U.S.C. 371		U.S. APPLICATION NO (If known, see 37 CFR 1.5) 09/979572
INTERNATIONAL APPLICATION NO. PCT/SG99/00043	INTERNATIONAL FILING DATE MAY 17, 1999	PRIORITY DATE CLAIMED May 17, 1999

TITLE OF INVENTION
EFFICIENT CODING IN PROCESSORS

APPLICANT(S) FOR DO/EO/US
MANISH BHARDWAJ and DEXTER CHIN

Applicant herewith submits to the United States Designated/Elected Office (DO/EO/US) the following items and other information:

1. [X] This is a **FIRST** submission of items concerning a filing under 35 U.S.C. 371.
2. [ ] This is a **SECOND** or **SUBSEQUENT** submission of items concerning a filing under 35 U.S.C. 371.
3. [X] This is an express request to begin national examination procedures (35 U.S.C. 371(f)). The submission must include items (5), (6), (9) and (21) indicated below.
4. [X] The US has been elected by the expiration of 19 months from the priority date (Article 31).
5. [ ] A copy of the International Application as filed (35 U.S.C. 371(c)(2))
   a. [X] is attached hereto (required only if not communicated by the International Bureau).
   b. [ ] has been communicated by the International Bureau.
   c. [ ] is not required, as the application was filed in the United States Receiving Office (RO/US).
6. [ ] An English language translation of the International Application as filed (35 U.S.C. 371(c)(2)).
   a [ ] is attached hereto.
   b. [ ] has been previously submitted under 35 U.S.C. 154(d)(4).
7. [X] Amendments to the claims of the International Aplication under PCT Article 19 (35 U.S.C. 371(c)(3))
   a. [ ] are attached hereto (required only if not communicated by the International Bureau).
   b. [ ] have been communicated by the International Bureau.
   c. [ ] have not been made, however, the time limit for making such amendments has NOT expired.
   d. [X] have not been made and will not be made
8. [ ] An English language translation of the amendments to the claims under PCT Article 19 (35 U.S.C. 371 (c)(3)).
9. [ ] An oath or declaration of the inventor(s) (35 U.S.C. 371(c)(4)).
10. [ ] An English lanugage translation of the annexes of the International Preliminary Examination Report under PCT Article 36 (35 U.S.C. 371(c)(5)).

Items 11 to 20 below concern document(s) or information included:

11. [ ] An Information Disclosure Statement under 37 CFR 1.97 and 1.98.
12. [ ] An assignment document for recording. A separate cover sheet in compliance with 37 CFR 3.28 and 3.31 is included.
13. [ ] A FIRST preliminary amendment.
14. [ ] A SECOND or SUBSEQUENT preliminary amendment.
15. [ ] A substitute specification
16. [ ] A change of power of attorney and/or address letter.
17. [ ] A computer-readable form of the sequence listing in accordance with PCT Rule 13ter.2 and 35 U.S.C. 1.821 - 1.825.
18. [ ] A second copy of the published international application under 35 U.S.C. 154(d)(4)
19. [ ] A second copy of the English language translation of the international application under 35 U.S.C. 154(d)(4).
20. [X] Other items or information:

EXPRESS MAIL CERTIFICATE

U.S. APPLICATION NO (if known, see 37 CFR 1.5)	INTERNATIONAL APPLICATION NO	ATTORNEY'S DOCKET NUMBER
09/979572	PCT/SG99/00043	1178.P002US

21 [X] The following fees are submitted

BASIC NATIONAL FEE (37 CFR 1.492 (a) (1) - (5)):	CALCULATIONS	PTO USE ONLY
Neither international preliminary examination fee (37 CFR 1.482) nor international search fee (37 CFR 1.445(a)(2)) paid to USPTO and International Search Report not prepared by the EPO or JPO	$1040.00	
International preliminary examination fee (37 CFR 1.482) not paid to USPTO but International Search Report prepared by the EPO or JPO	$890.00	
International preliminary examination fee (37 CFR 1.482) not paid to USPTO but international search fee (37 CFR 1.445(a)(2)) paid to USPTO	$740.00	
International preliminary examination fee (37 CFR 1.482) paid to USPTO but all claims did not satisfy provisions of PCT Article 33(1)-(4)	$710.00	
International preliminary examination fee (37 CFR 1.482) paid to USPTO and all claims satisfied provisions of PCT Article 33(1)-(4)	$100.00	
ENTER APPROPRIATE BASIC FEE AMOUNT =		$ 890
Surcharge of $130.00 for furnishing the oath or declaration later than [ ] 20 [ ] 30 months from the earliest claimed priority date (37 CFR 1.492(e)).		$

CLAIMS	NUMBER FILED	NUMBER EXTRA	RATE	$
Total claims	74 - 20 =	54	x $18.00	$ 972
Independent claims	3 - 3 =		x $84.00	$ 0
MULTIPLE DEPENDENT CLAIM(S) (if applicable)			+ $280.00	$ 280
TOTAL OF ABOVE CALCULATIONS =				$ 1252
[ ] Applicant claims small entity status. See 37 CFR 1.27 The fees indicated above are reduced by 1/2			+	$
SUBTOTAL =				$
Processing fee of $130.00 for furnishing the English translation later than [ ] 20 [ ] 30 months from the earliest claimed priority date (37 CFR 1.492(f)).				$
TOTAL NATIONAL FEE =				$ 2142
Fee for recording the enclosed assignment (37 CFR 1.21(h)). The assignment must be accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31) $40.00 per property +				$
TOTAL FEES ENCLOSED =				$ 2142
			Amount to be refunded:	$
			charged:	$

a. [X] A check in the amount of $ 2142 to cover the above fees is enclosed.

b. [ ] Please charge my Deposit Account No. _____ in the amount of $_____ to cover the above fees. A duplicate copy of this sheet is enclosed

c. [ ] The Commissioner is hereby authorized to charge any additional fees which may be required, or credit any overpayment to Deposit Account No. _____ A duplicate copy of this sheet is enclosed.

d. [ ] Fees are to be charged to a credit card. WARNING: Information on this form may become public. **Credit card information should not be included on this form.** Provide credit card information and authorization on PTO-2038.

NOTE: Where an appropriate time limit under 37 CFR 1.494 or 1.495 has not been met, a petition to revive (37 CFR 1.137 (a) or (b)) must be filed and granted to restore the application to pending status.

SEND ALL CORRESPONDENCE TO.

LAWRENCE N. GINSBERG
907 CITRUS PLACE
NEWPORT BEACH, CA 92660-3227
TEL/FAX 949-640-6261

SIGNATURE
LAWRENCE N. GINSBERG

NAME
30,943

REGISTRATION NUMBER

EFFICIENT CODING IN PROCESSORS

Field of the Invention

The invention generally relates to processors and, more particularly, to improving coding efficiency in processors.

Background of the Invention

In general, a digital computer performs tasks defined by a list of internally stored instructions. The list of instructions or computer codes, which is referred to as a program, is stored in the computer's main memory. To perform a given computational task, the central processing unit or processor of the digital computer fetches individual instructions of the program for execution. A program counter within the computer points to the next instruction to be executed.

The length of the instruction or instruction word depends on the specific computer design. Fig. 1 shows a typical instruction word 101. The instruction word includes an opcode field 110 and one or more operand fields 115, depending on the instruction. The opcode defines the operation and the operand or operands may

contain data that are to be operated by the given instruction.

Various types of computer architectures exist, such as Von-Neumann and the Harvard Architectures. See generally Hennessy and Pallerson, Computer Architecture: A Quantitative Approach (1996), which is herein incorporated by reference for all purposes. In the Von-Neumann Architecture, both data and instructions are transmitted between the processor and memory by a bi-directional bus. As such, the instruction stream includes both data and instructions. In contrast, instructions and data are transmitted on separate buses in the Harvard Architecture, separating the data from the instruction stream.

An instruction requiring an operand that is a constant known at compile time is referred to as an immediate instruction. The known operand is referred to as an immediate data. Immediate data often results in situations where an instruction is operating on a constant or where the operand is an address pointer.

In all current processors, regardless of the architecture, immediate data is included as part of the instruction word (stored in the operand field 115 of the instruction word). That is, the immediate data is part

of the instruction stream. In some instances, the length of the immediate data can exceed the length of the operand field of the instruction word. Immediate data which exceeds the length of the operand field is typically referred to as a long immediate data. To accommodate the long immediate data, an extension 220 is provided in the instruction word 101, as shown in Fig. 2. The instruction extension is essentially an instruction word containing the immediate data that follows the instruction word 101. As such, an additional cycle is required to handle the extension of a long immediate data.

The presence of extensions can complicate decoding and pipelining issues. To avoid extensions, either constraints are imposed to limit the length of immediate data or the instruction word comprises an operand field having a sufficient length to handle the widest possible immediate data. In the first case, imposing constraints on the length of immediate data may not be desirable due to application requirements. For example, in digital signal processing (DSP) applications, the length of the immediate data is often governed by domain characteristics such as signal-to-noise (SNR) requirements. Likewise, increasing the length of the

instruction word to handle the long immediate data may be undesirable as this leads to inefficient use of memory, particularly with embedded systems where chip size is a major concern.

As evidenced by the foregoing discussion, it is desirable to improve coding efficiency without imposing constraints on the length immediate data.

SUMMARY OF THE INVENTION

In one embodiment, the processor includes a program counter which stores an instruction address. The instruction at the instruction address is fetched by the processor and decoded. If the decoded instruction comprises an immediate instruction, the corresponding immediate data is fetched from a data table.

In one embodiment, immediate data is stored in the data table in the order determined by a flow analysis. The flow analysis identifies immediate instructions and stores the corresponding immediate data in the program order of the immediate instructions. In one embodiment, the flow analysis identifies immediate instructions and stores the corresponding immediate data in substantially the program order of the immediate instructions, taking branching into account. The information provided by the

flow analysis is used to generate instruction coding to enable retrieval of the immediate data from the data table.

DESCRIPTION OF THE DRAWINGS

Fig. 1 shows an instruction word;

Fig. 2 shows an immediate instruction word which includes an instruction extension;

Fig. 3 shows a static flow analysis in accordance with one embodiment of the invention;

Fig. 4 shows a static flow analysis in accordance with another embodiment of the invention;

Figs. 5-6 illustrate the concept of outermost sections in accordance with the invention;

Fig. 7 shows the placement of immediate data from the program into a DT in accordance with one embodiment of the invention;

Fig. 8 illustrates a flow analysis in accordance with another embodiment of the invention;

Fig. 9 shows a process for operating a processor in accordance with one embodiment of the invention;

Fig. 10 shows a processor in accordance with one embodiment of the invention;

Fig. 11 illustrates the instruction coding in

accordance with one embodiment of the invention;

Fig. 12 shows a process for operating a processor in accordance with another embodiment of the invention;

Fig. 13 shows a processor in accordance with another embodiment of the invention; and

Fig. 14 shows a processor in accordance with yet another embodiment of the invention.

DESCRIPTION OF THE INVENTION

The present invention relates generally to a coding scheme for handling immediate data in a computer program, which makes efficient use of memory without imposing programming constraints. A processor, such as a microprocessor, a microcontroller, a digital signal processor (DSP), or an embedded system, executes the program. The present invention is particularly useful for handling immediate data in the processor's firmware.

The computer program can be generated using high-level computer languages, such as C++. Other high level languages are also useful. The program is processed to translate it into machine code usable by the processor. The program may be processed in several stages before it is finally translated into machine code. For example, most compilers initially process the program to generate

assembly code. An assembler then translates the code into machine code which is then stored into memory for use by the processor.

In accordance with the invention, a coding scheme that separates immediate data from the instruction stream is disclosed. As used herein, the term "immediate data" can refer to long immediate data or immediate data in general. Such a coding scheme eliminates instruction extensions without increasing the length of the instruction word or imposing constraints on programming, thus improving coding density and processing efficiency.

In one embodiment, the separation of immediate data from the instruction stream is facilitated by a static control flow analysis to identify and determine an order of the immediate data in the program. Immediate data are stored in a data table in the order determined by the flow analysis. The coding of an immediate instruction can include addressing information for retrieval of the immediate data by the processor when the immediate instruction is executed.

A post-processor, for example, performs the flow analysis. The post-processor can be incorporated as a part of a compiler, an assembler, or a tool used to

translate higher level codes into lower or machine level codes. Providing a post-processor which is separate from the tools used to translate higher level codes into lower level codes is also useful.

Fig. 3 shows a flow analysis in accordance with one embodiment of the invention. The flow analysis facilitates the separation of immediate data from the instruction stream. The flow analysis, for example, creates a model of a data table comprising immediate data of the immediate instructions from a program.

In one embodiment, the flow analysis identifies an immediate instruction in the program at step 310. At step 320, the immediate data of the immediate instruction is assigned a location in a separate storage that is accessible by the processor. The assignment of location can be explicit (e.g., assigning a specific location) or implicit (e.g., by placement of immediate data in the model). The separate storage, for example, comprises a read only memory (ROM) table or other memory storage locations. The separate storage is referred to as a data table (DT). The flow analysis also generates addressing information to enable retrieval of the immediate data when the processor executes the instruction. At step 330, the flow analysis continues

the process of identifying immediate data and assigning a location in the DT until all immediate data in the program are identified. At step 340, the flow analysis is completed once all immediate data have been identified.

The information generated by the flow analysis, such as the model of the DT, is used to replicate the actual DT. For example, the flow analysis provides the information that enables the storing or burning (for ROM) of the immediate data into the DT at their assigned addresses during chip design or fabrication. The flow analysis can also provide the function of creating the DT.

The information provided by the flow analysis can also be used to generate the instruction coding to retrieve the immediate data from the DT. The post-processor that performs the flow analysis can provide the function of generating the instruction coding. A separate post-processor that provides the instruction coding can also be useful. In a preferred embodiment, the flow analysis identifies the immediate data in their program execution order (program order) and assigns DT addresses to the immediate data to reflect their program order. In one embodiment, the flow analysis provides a

systematic approach to identify immediate data in substantially their program order, taking into account of branches, i.e., conditionally executed instructions.

Fig. 4 shows another embodiment of the static flow analysis. As shown, the flow analysis commences at step 410 by grouping the program codes into outermost sections. At step 420, the flow analysis identifies immediate data within an outermost section and assigns corresponding locations in the DT at step 430. The flow analysis also generates addressing information to enable retrieval of the immediate data when the processor executes the instruction. At step 440, the flow analysis determines whether there are more outermost sections that need to be analyzed. If there are more outermost sections that need to be analyzed, the flow analysis returns to step 420. Otherwise, the flow analysis finishes at step 450.

To understand the concept of outermost sections, the term section is first defined. A typical program comprises a plurality of codes and branches. A section refers to an atomic unit of the program and is defined as one or more instructions or codes in a program path between two branch nodes or points. A branch node or point can refer to either the start or the end of a

branch. An unconditional section refers to a section whose codes are all unconditionally executed during program execution. On the other hand, a section whose codes may not be executed during program execution is referred to as a conditional section.

An outermost section is defined as either an unconditional section or a group of conditional sections between corresponding start and end branch nodes that are unconditionally encountered during program execution. An outermost section that comprises an unconditional section is referred to as an unconditional outermost section. An outermost section that comprises a group of conditional sections is referred to as a conditional outermost section. From the definitions provided, a conditional outermost section comprises at least two conditional sections. In some instances, a conditional outermost section can comprise a plurality of conditional sections having numerous branches therein.

Figs. 5-6 show a sample program 501 to facilitate understanding of the static flow analysis in accordance with one embodiment of the invention. Referring to Fig. 5, the program includes a plurality of codes and branches. A start of a branch is represented with

arrows splitting off into two different paths of the program, and an end of a branch is represented with arrows merging into one path. The codes are grouped into sections 510, 521, 522, 530, 541-547, 551-554, and 560. A dashed line within a section represents an immediate instruction.

Referring to Fig. 6, the program 501 is grouped into outermost sections. Applying the rules pertaining to outermost sections, sections 510, 530, and 560 are categorized as outermost sections since they are unconditional sections. Sections 521 and 522 are grouped as an outermost section 610 because they are located between corresponding unconditional start and end branch points 624 and 625. Likewise, conditional sections 541-547 and 551-554 are grouped as an outermost section 640 since those conditional sections are located between corresponding unconditional start and end branch points 650 and 651.

In an embodiment of the invention, the static flow analysis identifies immediate data within an outermost section one outermost section at a time. In a preferred embodiment, the flow analysis identifies immediate data from the outermost sections in the program order of the outermost sections. The immediate data are assigned DT

addresses to reflect the outermost sections' program order. More preferably, immediate data within an outermost section are identified in their program order and assigned DT addresses to reflect their program order within the outermost section. Analyzing outermost sections in other orders and assigning addresses to reflect other orders can also be useful.

Fig. 7 illustrates the placement of immediate data from the program 501 into a DT table 701 in accordance with one embodiment of the invention. As shown, the program is grouped into outermost sections. The program order of the outermost sections is 510, 610, 530, 640, 560. Since outermost section 510 is first in the program order, the immediate data corresponding to the immediate instructions therein are assigned and stored in addresses of the first region 710 of the DT. Subsequent outermost sections 610, 530, 640, and 560 occupy subsequent regions 720, 730, 740, and 750, respectively, of the DT.

As previously described, an unconditional outermost section comprises more than one conditional sections. In one embodiment of the invention, the flow analysis handles a conditional outermost section by separating it into right and left subsections. One subsection is

analyzed first before analyzing the other. In one embodiment, the flow analysis analyzes the right subsection first and then the left subsection. Immediate data in the right subsection are identified and assigned DT addresses first and then immediate data in the left subsection are identified and assigned DT addresses. The DT addresses preferably reflect the program order of the immediate data within the subsection. Analyzing the left subsection first and then the right subsection is also useful. In a preferred embodiment, the order in which the subsection is analyzed is consistent for all the outermost sections.

A subsection of an outermost section can comprise a plurality of conditional sections. In such a case, the subsection can be grouped into outermost sections as viewed within the subsection. The immediate data within an outermost section preferably are identified and assigned DT addresses to reflect their program order within the outermost sections, as previously described. In this manner, the program is recursively analyzed to construct the DT. Thus, the immediate data are analyzed and assigned DT addresses to reflect substantially their program order, taking branching into account.

Fig. 8 illustrates the recursive construction of DT by the flow analysis. Similar to Fig. 7, the program 501 comprises outermost sections having a program order of 510, 610, 530, 640, and 560. The immediate data of the outermost sections are assigned DT addresses to reflect the program order of the outermost sections. Immediate data from outermost sections 510, 610, 530, 640, and 560 are assigned addresses in DT regions 710, 720, 730, 740, and 750, respectively.

Outermost section 610, which comprises sections 521 and 522, is separated into right and left subsections. Sections 521 and 522 can be viewed as outermost sections within their respective subsections (i.e., they unconditional sections within their respective subsections). In accordance with one embodiment, the right section is analyzed before the left section, resulting in the storing of immediate data from the right subsection in portion 822 of DT region 720 and the storing of immediate data from the left subsection in portion 823.

Outermost section 640 is also separated into right and left subsections 810 and 850 and their immediate data are stored in portions 862 and 863, respectively, of the DT region 740.

The right subsection comprises a plurality of sections which can be grouped into outermost sections with respect to the subsection. Sections 541 and 547 are outermost sections since they are considered unconditional sections; sections 542-546 are grouped as an outermost section 830 since they are located between corresponding unconditional start and end branch points 821 and 822. Immediate data from outermost sections 541, 830,and 547 of the right subsection are stored in portions 870, 883, and 876 of DT region 740.

Outermost section 830 can be further separated into right and left subsections. As such, the analysis is recursively applied to the program code to group the code into outermost sections and separate them into right and left subsections. The recursive analysis, in accordance with one embodiment of the invention, results in assigning DT addresses for the immediate data from sections 542, 543, 544, 545, and 546 in subportions 871, 872, 873, 874, and 875 of the portion 883.

After immediate data from the right subsection 810 have been assigned DT addresses, the left subsection 850 is analyzed. The analysis, which is similar to that of the right subsection, groups the codes into outermost sections 551, 840, and 554. The immediate data from the

outermost sections are assigned DT addresses in portion 863 of DT region 740 to reflect their program order. In particular, immediate data from outermost section 551 are stored in portion 877, immediate data from outermost section 840 are stored in portion 885, and immediate data from outermost section 554 are stored in portion 880. Outermost section 840 can be separated into right and left subsections. The right subsection comprises section 552 and the left subsection comprises section 553. Immediate data from the right subsection are stored in subportion 878 and immediate data from the left subsection are stored in subportion 879.

In one embodiment, a subroutine *Build_DT* recursively analyzes the program and creates a model of a DT with its contents. The information produced by the *Build_DT* routine can be, for example, provided to another tool to create the DT. An example of the subroutine *Build_DT* is as follows:

```
Build_DT(section, total_outermost_sections)
for i = 0 ... total_outermost_sections-1)
        Place(outersection(i));
endfor
end Build_DT
```

As shown, *Build_DT* routine receives parameters section and total_outermost_sections. The section parameter represents the section to be placed (which is initially the whole program), and the total_outermost_sections parameter represents the number of outermost sections within the program. The routine analyzes the outermost sections in the program order of the outermost sections, starting from the first one (i=0) to the last one (total_outermost_section-1). For each outermost section, *Build_DT* calls a routine *Place* to store the immediate data into the DT. In one embodiment, *Place* stores the immediate data into the DT in their program order within the outermost section.

An example of the *Place* subroutine is as follows:

```
Place(section)
if section is unconditional
      place all the immediates required by routine
      in their data order in the DT.
else
      Build_DT(section.right,
            get_outermost(section.right))
      Build_DT(section.left,
            get_outermost(section.left))
endif
```

end Place

The subroutine *Place* analyzes the outermost section to determine whether it is a conditional or an unconditional outermost section. An unconditional outermost section indicates that it contains one section. If an unconditional outermost section is detected, the routine stores the section's immediate data in the DT. The immediate data are stored in their program order or in the order in which they are identified. Once all immediate data within the outermost section are stored in the DT, the *Place* routine terminates and returns to the *Build_DT* routine to continue analyzing the next outermost section, if any.

A conditional outermost section, on the other hand, indicates that there exists a plurality of sections. When a conditional outermost section is encountered, the *Place* routine separates the outermost section into right and left subsections and analyzes each one individually. Illustratively, the *Place* routine analyzes the right subsection and then the left subsection.

The *Place* routine calls the *Build_DT* routine to analyze the subsection. Simultaneously, a routine get_outermost is invoked to find the outermost sections

within the subsection. As such, *Place* views each subsection as if it were a separate program and analyzes accordingly. In this manner, the post-processor recursively analyzes the program to construct the DT model.

The flow analysis also provides information to generate the instruction coding for handling the separation of immediate data from the instruction stream. In accordance with the invention, the flow analysis provides addressing information related to the immediate data in the DT. Such addressing information can be in the form of relative or absolute addressing information. For relative addressing, the flow analysis may perform more than one pass to obtain the addressing information. As an example, a first pass may be used to compute the sections and subsections while a second pass is used to assign the offset for relative addressing of the immediate data.

The addressing information generated by the flow analysis can be used by, for example, a compiler or an assembler to generate the instruction coding to accommodate the use of a DT. In one embodiment, the instruction coding informs the processor that the current instruction needs an immediate data as well as

addressing information to retrieve the immediate data. Indicating that an immediate data is needed can be achieved by, for example, providing a bit in the instruction to serve as a flag to indicate that an immediate data is needed. Alternatively, the processor could be informed implicitly by embedding the request for immediate data in the opcode. The addressing information identifying the location of the immediate data can be stored in the operand field of the instruction word. The information is transferred to and used by the DT pointer to access the DT for retrieving the immediate data. In one embodiment, the addressing information contains absolute addressing.

Fig. 9 shows a process for operating a processor in accordance with one embodiment of the invention. The processor fetches an instruction at step 910. For example, at the beginning of program execution, the program counter is reset to 0 to point to the first instruction. The processor fetches the instruction to which the program counter points. At step 920, the processor determines whether the instruction is an immediate instruction or not. If it is not an immediate instruction, the processor proceeds to step 950 for instruction execution. If the instruction is an

immediate instruction, the addressing information stored in the operand field is loaded into a DT pointer at step 930. At step 940, the processor retrieves the immediate data in the DT at the address provided by the DT pointer and executes the instruction at step 950. At step 960, the processor determines if the program is completed. If the program is not completed, the next instruction is fetched at step 910. The process continues until all instructions have been executed, terminating the program at step 970.

Fig. 10 shows a processor 10 in accordance with one embodiment of the invention. The processor can comprise a Von-Neuman or Harvard architecture. Other processor architectures are also useful. As shown, the processor comprises a program counter 30. The program counter is coupled to the memory (not shown) which stores the program. The program pointer points to the instruction that is to be fetched for execution. The instruction is fetched from memory and stored in the instruction register (IR) 35. An instruction decoder (ID) 31 decodes the instruction in the IR. If the ID determines that it is an immediate instruction, it activates a DT pointer 60 and a DT 50, causing the addressing information stored in the operand field of the immediate

instruction word to be loaded into the DT pointer. The immediate data at the address contained in the DT pointer is made available to the processor for executing the immediate instruction. Alternatively, the addressing information can be used directly to access the DT, avoiding the need of a DT pointer.

In another embodiment of the invention, the coding uses relative addressing information. The relative addressing information comprises an offset that is used to address the immediate data in the DT. In one embodiment, the offset is with respect to the DT address of the immediate data from the previously executed immediate instruction. Employing the use of relative addressing advantageously reduces the number of bits needed to address the DT, particular for applications with a very large number of DT entries.

In accordance with one embodiment of the invention, the DT pointer is reset to a value that points to the first entry in the DT at the end of program execution or prior to the commencement of program execution. For example, the DT pointer is reset to zero.

The offset modifies the DT pointer for addressing the DT. In one embodiment, the offset is added to the current value stored in the DT pointer. To accommodate

branching in the program, a pre/post bit is provided to indicate whether a pre-offset or post-offset format is used. This results in an instruction coding in which the operand field of an immediate instruction word comprises the format (pre/post, #offset). In one embodiment, a pre/post bit having a value of 1 represents a pre-offset format, and a 0 value represents a post-offset format. In the pre-offset, the offset is added to the current value of the DT pointer (current value + offset) before retrieving the immediate data from the DT. After the immediate data is retrieved, the value in the DT is incremented to point to the DT address containing the immediate data of the next immediate instruction. For example, the DT printer is incremented by 1 (current value + offset + 1). In the post-offset format, the current value of the DT pointer (current value) is used to retrieve immediate data from the DT. The DT pointer is incremented, for example, by 1 (current value + 1) after the immediate data is retrieved to point to the immediate data of the next immediate instruction.

In accordance with one embodiment of the invention, the coding of the pre/post bit and offset value depends on whether the section is conditional or unconditional.

For unconditional sections, the immediate instructions have a coding wherein the pre/post bit indicates that the offset is added after instruction execution (post) and the offset value (#offset) is 1. The term "unconditional sections" also refers to conditional sections which are viewed as unconditional sections within a subsection.

As for conditional sections, the last immediate instruction of a right subsection has a coding format in which the pre/post bit indicates that the offset is added after instruction execution and the offset value is equal to the number of immediate instructions on the left subsection + 1. The first instruction of the left subsection has a coding in which the pre/post bit indicates that the offset is added before instruction execution (pre) and the offset value is equal to the number of immediate instructions in the right subsection. All other immediate instructions have pre/post bit equal to post and offset equal to 1. The last immediate instruction in the program is always coded to reset the DT pointer to zero (i.e., beginning of the DT). This can be done by setting the offset value to zero while the value of pre/post bit has no effect (don't care).

As described, the maximum value (m) of the offset within a program in the case where the right subsection is analyzed first is equal either to the maximum number of immediate instructions in a right outermost subsection or to the number of immediate instructions in the left outermost subsection + 1, whichever is greater. The number of bits needed to code the offset in the immediate instructions is equal to $\lceil \log_2(m+1) \rceil$. The total number of bits needed in the operand field to code an immediate instruction is equal to $1 + \lceil \log_2(m+2) \rceil$, where the additional bit is for the pre/post flag. In some cases, the coding may include an additional bit to explicitly instruct the processor that the current instruction is an immediate instruction. In most embedded applications, especially in data-intensive DSP, the value of m is relatively small, such as less than 8. For m=8, the number of operand bits required is at most 5 bits (4 for the offset, 1 for the pre/post). In the case where an immediate instruction is identified explicitly by the use of a bit, only an additional bit is needed. This can significantly reduce the number of bits needed in the operand field to handle immediate data and eliminate the need for instruction extensions, resulting in more efficient coding.

Fig. 11 illustrates the coding in accordance with one embodiment of the invention. As shown, the program 501 is grouped into outermost sections 510, 610, 530, 640, and 560. A dashed box indicates an atomic unit or section. An immediate instruction within the section is represented by a number whose value indicates its program order within the section. The coding for the immediate instruction is enclosed in brackets. A post or pre indicates that the offset is added after or before instruction execution, and the value after # indicates the offset. An alternating short and long dashed box represents a right or left subsection. The number on top of each subsection reflects the number of immediate instructions within the subsection.

For unconditional outermost sections 510 and 530, the immediate instructions are coded (post, #1). The last immediate instruction of the program, which is located in outermost section 560, is coded (pre/post, #0) to reset the DT pointer to zero. Outermost section 610 is separated into right and left subsections. The right subsection comprises section 521 and the left subsection comprises section 522. The last immediate instruction in the right subsection is coded (post, #8) to provide an offset of 8 (7 instructions on the left

subsection + 1). The first instruction on the left section is coded (pre, #6).

As for outermost section 640, the right and left subsections 810 and 850 each comprises a plurality of sections. The sections within each subsection are recursively analyzed, grouping the sections into outermost sections and separating the outermost sections into right and left subsections, if necessary. For example, the right subsection 810 is grouped into outermost sections 541, 830, and 547. Outermost section 830 can further be separated into right and left subsections 110 and 150. The immediate instructions are coded as previously described.

Fig. 12 shows a process for operating a processor in accordance with another embodiment of the invention. At the beginning of program execution at step 210, the DT pointer is reset to zero to point to the first entry in the DT. The program counter is also reset to point to the first instruction. At step 220, the processor fetches the instruction to which the program counter points. At step 230, the processor determines whether the instruction is an immediate instruction or not. If it is not an immediate instruction, the processor proceeds to step 280 for instruction execution. If the

instruction is an immediate instruction, the processor further determines if the offset information stored in the operand field is pre-format or post-format at step 240. Also, the processor can analyze the offset to see if it is equal to zero or not, indicating whether or not there are any more immediate instructions. A zero offset resets the DT pointer to zero. If the offset is post-format coded, the process proceeds to step 250. At step 250, the current value in the DT pointer is used to retrieve the immediate data in the DT. The instruction is executed at step 255. After instruction execution, the offset is added to the current value in the DT pointer and stored therein at step 252. If the offset is pre-format coded, the process proceeds to step 260. At step 260, the offset is added to the current value of the DT pointer and the new value is used to retrieve the immediate data in the DT at step 261. At step 262, the processor executes the immediate instruction. The DT pointer is then incremented by 1 at step 263. At step 290, the processor determines if there are anymore instructions. If there are, the next instruction is fetched at step 220. The process continues until all instructions have been executed, terminating the program at step 295.

Fig. 13 shows a processor 310 in accordance with one embodiment of the invention. As shown, the processor comprises a program counter 330. The program counter is coupled to the memory (not shown) which stores the program. The program pointer points to an instruction. The instruction is fetched from memory and stored in an IR 335. An ID 331 decodes the instruction in the IR. If the ID determines that it is an immediate instruction, it activates a DT addressing unit 360 and DT 350. The DT addressing unit comprises a DT pointer 361, which receives the addressing information from the instruction. The addressing information in the DT pointer is used to fetch the immediate data from the DT for use by the processor.

The immediate instruction provides relative addressing information to identify the location of the immediate data in the DT. In one embodiment, the relative addressing information includes an offset and a pre/post bit. If the pre/post bit represents a post-offset, the DT addressing unit uses the current value in the DT pointer to address the DT to retrieve the immediate data. The offset is added to the current value of the DT pointer after the immediate data is fetched. If the pre/post bit represents a pre-offset,

the DT addressing unit adds the offset to the current value of the DT pointer. The new value in the DT pointer is used to fetch the immediate data. After the immediate data is fetched, the DT is incremented by 1 to point to another address containing the immediate data of, for example, the next immediate instruction. As shown, a pre/post value of 1 represents pre-offset, and a pre/post value of 0 represents post-offset.

In one embodiment, the addressing unit comprises adders 363 and 364, the DT pointer, and a multiplexer 368. Adder 363 receives as inputs the offset value from the IR and the current value in the DT pointer. The output of adder 363 is coupled to adder 364. Also coupled to adder 364 is the pre/post bit. The output of adder 364 is coupled to the DT pointer, whose output is coupled to the multiplexer 368. The pre/post bit also serves as a select signal for the multiplexer.

In operation, adder 363 adds the offset to the current value in the DT pointer. The result of adder 363 is passed to both the adder 364 and to the multiplexer 368. A pre/post bit equal to 1 indicates a pre-offset addressing of the DT, causing the multiplexer to select the sum of the offset and current value of the DT pointer (current DT printer + offset) as the DT

address containing the immediate data. Adder 364 then adds the results of adder 363 to the pre/post bit, which is 1. The result (offset + current DT pointer + 1) is then loaded into the DT pointer for the next DT access.

A pre/post bit equal to 0 indicates a post-offset addressing of the DT. This causes the multiplexer to select the value of the current DT pointer to access the DT. After the immediate data is fetched from the DT, adder 363 adds the current DT value to the offset (current DT pointer + offset) whose result is then added by adder 364 to the pre/post bit, which is equal to 0. The result of adder 364 (current DT pointer + offset) is then loaded into the DT pointer to use for the next DT access.

In the case where the program does not include branching, the pre/post bit can be eliminated. Additionally, the instruction coding need only identify that the instruction is an immediate instruction. This is because the offset is always 1 if the immediate data is stored in their program order. By creating a scenario in which the offset is constant for all immediate instructions, the DT addressing unit can be configured to provide the offset. Thus, the need for offset information in the instruction coding can be

avoided.

Fig. 14 shows an embodiment of a processor for handing coding without branching. As shown, the processor 410 comprises a program counter 430. The program counter is coupled to the memory (not shown) which stores the program. The program pointer points to an instruction that is to be executed. The instruction is fetched from memory and stored in the IR 435. An ID 431 decodes the instruction in the IR. If the ID determines that it is an immediate instruction, it activates a DT addressing unit 460 and a DT 450. The immediate data stored in the location referenced by a DT pointer 464 in the DT addressing unit is fetched for the processor's use. After the immediate data is fetched, the current value in the DT pointer is incremented by 1 using an adder 465 in the DT addressing unit.

While the invention has been particularly shown and described with reference to various embodiments, it will be recognized by those skilled in the art that modifications and changes may be made to the present invention without departing from its scope. The scope of the invention should therefore be determined not with reference to the above description but with reference to the appended claims along with their full scope of

equivalents.

what is claimed is:-

1. A processor comprising:

a data table for storing immediate data of immediate instructions;

a program counter for storing an instruction address of an instruction, wherein the processor fetches the instruction from the instruction address in the program counter during program execution; and

an instruction decoder for decoding the instruction fetched by the processor, wherein an immediate data from the data table is provided to the processor if the instruction is an immediate instruction.

2. The processor of claim 1 further comprises an instruction register coupled to the instruction decoder, the instruction register stores the instruction fetched by the processor during program execution and passes the instruction to the instruction decoder for decoding.

3. The processor of claim 2 wherein the instruction register is coupled to the data table, the instruction register provides an address of the immediate data in

the data table when the immediate instruction is decoded by the instruction decoder.

4. The processor of claim 2 further comprises a data table addressing unit coupled to the instruction decoder and data table, the data addressing unit providing an address of the immediate data in the data table.

5. The processor of claim 4 wherein the data table addressing unit comprises a data table pointer for storing the address.

6. The processor of claim 4 wherein the instruction register is coupled to the data table addressing unit, the instruction register passes addressing information to the instruction addressing unit to provide the address when the immediate instruction is decoded by the instruction decoder.

7. The processor of claim 6 wherein the data table addressing unit comprises a data table pointer for storing the addressing information which serves as the address.

8. The processor of claim 1 further comprises a data table addressing unit coupled to the instruction decoder and data table, the data addressing unit providing an address of the immediate data in the data table.

9. The processor of claim 8 wherein the data table addressing unit comprises a data table pointer for storing the address.

10. The processor of claim 8 further comprises an instruction register coupled to the instruction decoder, the instruction register stores the instruction fetched by the processor during program execution and passes the instruction to the instruction decoder for decoding.

11. The processor of claim 10 wherein the instruction register is coupled to the data table addressing unit, the instruction register passes addressing information to the instruction addressing unit to provide the address when the immediate instruction is decoded by the instruction decoder.

12 The processor of claim 11 wherein the data table

addressing unit comprises a data table pointer for storing the addressing information which serves as the address.

13. The processor of claim 1 further comprises a data table addressing unit coupled to the instruction decoder and the data table, the data addressing unit provides an address of the immediate data in the data table when the instruction decoder decodes the immediate instruction.

14. The processor of claim 13 wherein the data table addressing unit comprises an incrementor, the incrementor increments the address after the immediate data is provided to produce a new address for a new immediate instruction.

15. The processor of claim 13 wherein the data table addressing unit comprises:

a data table pointer for storing the address; and

an incrementor, the incrementor increments the address in the data table pointer after the immediate data is provided to produce a new address in the data table pointer for a next immediate instruction executed by the processor.

16. The processor of claim 15 wherein the incrmentor comprises an adder, the adder is coupled to the data table pointer, the adder adds the address in the table pointer and an index to produce the new address.

17. The processor of claim 16 wherein the index comprises a 1.

18. The processor of claim 13 wherein an addressing information is passed to the addressing unit, the addressing information comprises an index for indexing the address to produce a new address of another immediate data in the data table for another immediate instruction.

19. The processor of claim 13 wherein addressing information is passed to the addressing unit, the addressing information comprises an index for indexing the address to produce a new address pointing to a next immediate data in the data table for a next immediate instruction fetched by the processor.

20. The processor of claim 19 further comprises an instruction register coupled to the instruction decoder

and the data table addressing unit, the instruction register stores the instruction fetched by the processor during program execution and passes the instruction to the instruction decoder for decoding, when the decoder decodes the immediate instruction, the instruction register passes the addressing information contained in the instruction to the data addressing unit.

21. The processor of claim 20 wherein the data table addressing unit comprises:

a data table pointer for storing the address; and

an adder for adding the addressing information to the address after the immediate data is provided to produce a new address in the data table pointer for a next immediate instruction executed by the processor.

22. The processor of claim 1 further comprises:

an instruction register coupled to the instruction decoder, the instruction register stores the instruction fetched by the processor during program execution and passes the instruction to the instruction decoder for decoding; and

a data table addressing unit coupled to the data table and instruction register, the instruction register

passing relative addressing information contained in the instruction to the data table addressing unit when the decoder decodes the immediate instruction, the relative addressing information, which comprises an index and a format indicator, is used to provide an address of the immediate data in the data table.

23. The processor of claim 22 wherein the data addressing unit comprises a data table pointer containing a value,

if the format indicator comprises a post-format, the value serves as the address and after the immediate data is provided to the processor, the index is added to the value to produce a new value in the data table pointer for a next immediate instruction, and

if the format indicator comprises a pre-format, the index is added to the value to produce the address to the immediate data and the address is incremented by 1 after the immediate data is provided to the processor to produce a new value in the data table pointer for the next immediate instruction.

24. The processor of claim 23 wherein the format indicator comprises a binary bit having a logic 1 and

logic 0 value, the logic 1 indicating a pre-format and the logic 0 indicating the post-format.

25. The processor of claim 24 wherein the addressing unit comprises:

a first adder comprising a first input coupled to the instruction register for receiving the offset and a second input coupled to an output of the data table pointer for receiving the value contained therein;

a second adder comprising a first input coupled to the instruction register for receiving the format indicator, a second input coupled to an output of the first adder, and an output coupled to the data table pointer; and

a multiplexor comprising a first input coupled to an output of the data table pointer, a second input coupled to the output of the first adder, and a select input coupled to the instruction register for receiving the format indicator to select an multiplexor output from the first and second multiplexor inputs.

26. The processor of claim 1 wherein the data table comprises immediate data stored in an order determined by a static flow analysis to identify immediate

instructions within a program.

27. The processor of claim 26 wherein an immediate instruction comprises addressing information to enable the processor to retrieve a corresponding immediate data to the immediate instruction.

28. The processor of claim 27 wherein the addressing information comprises absolute addressing information.

29. The processor of claim 27 wherein the addressing information comprises relative addressing information.

30. The processor of claim 29 further comprises data addressing unit coupled to the data table, the data addressing unit receives the relative addressing information and produces an address to the corresponding immediate data in the data table.

31. A method of executing instructions in a processor, the method comprises separating immediate data of immediate instructions from an instruction stream.

32. The method recited in claim 1 wherein separating the immediate data from the instruction stream comprises:

fetching an instruction from a program;

decoding the instruction to determine a type of instruction; and

if the instruction comprises an immediate instruction, fetching an immediate data corresponding to the immediate instruction from a data table.

33. The method recited in claim 32 wherein the immediate data of the immediate instructions are stored in the data table in an order determined by a flow analysis for identifying the immediate instructions in a program.

34. The method of claim 32 or 33 further comprises providing addressing information for fetching the immediate data from the data table.

35. The method of claim 34 wherein the immediate instruction provides the addressing information.

36. The method of claim 35 further comprises storing the addressing information in a data table pointer.

37. The method of claim 35 wherein the addressing information comprises relative addressing information.

38. The method of claim 37 further comprises processing the relative addressing information by a data table addressing unit to generate an address for fetching the immediate data from the data table.

39. The method of claim 38 wherein processing the relative addressing information comprises:

using a value stored in a data pointer of the data addressing unit to serve as the address; and

after fetching the immediate data, adding the relative addressing information to the value to produce a next address in the data table for fetching a next immediate data for a next immediate instruction.

40. The method of claim 38 wherein the addressing information comprises an index and a format indicator.

41. The method of claim 40 wherein processing the relative addressing information comprises:

if the format indicate comprises a post-format,

using a value stored in a data pointer of the data addressing unit to serve as the address, and

after fetching the immediate data, adding the index to the address to produce a next address in the data table for fetching a next immediate data for a next immediate instruction; and

if the format indicator comprises a pre-format,

adding the index to the value to serve as the address for fetching the immediate data in the data table, and

incrementing the address after the immediate data is fetched to produce a next address in the data table for fetching a next immediate data for a next immediate instruction.

42. The method of claim 34 wherein the addressing information comprises relative addressing information.

43. The method of claim 42 wherein providing the relative addressing information comprises:

using a value in a data pointer to provide the addressing information to fetch the immediate data from the data table; and

incrementing the value to provide a new addressing information for fetching another immediate data for a subsequent immediate instruction fetched by the processor.

44. A method of facilitating the separation of immediate data of immediate instructions from an instruction stream, the method comprises performing a flow analysis to identify the immediate instructions within a program.

45. The method of claim 44 further comprises assigning data table locations for immediate data of the immediate instructions.

46. The method of claim 45 further comprises storing immediate data of the immediate instructions in the assigned data table locations.

47. The method of claim 44 wherein the flow analysis identifies the immediate instructions in a program order of the immediate instructions.

48. The method of claim 47 further comprises assigning data table locations for immediate data in the program order of the immediate instructions.

49. The method of claim 45 further comprises storing immediate data of the immediate instructions in the assigned data table locations.

50. The method of claim 44 wherein the flow analysis identifies the immediate instructions in substantially a program order of the immediate instructions taking branching into account.

51. The method of claim 50 further comprises assigning data table locations for immediate data in substantially the program order of the immediate instructions.

52. The method of claim 51 further comprises storing immediate data of the immediate instructions in the assigned data table locations.

53. The method of claim 44 wherein identifying immediate data comprises:

grouping the program into outermost sections; and

identifying immediate data within the outermost sections.

54. The method of claim 53 further comprises assigning data table locations for immediate data of the immediate instructions.

55. The method of claim 54 further comprises storing immediate data of the immediate instructions in the assigned data table locations.

56. The method of claim 53 wherein identifying immediate data within the outermost sections comprises identifying immediate data within an outermost section in a program order of the outermost sections.

57. The method of claim 56 further comprises assigning data table locations for immediate data of the immediate data in the program order of the outermost sections.

58. The method of claim 57 further comprises storing immediate data of the immediate instructions in the assigned data table locations.

59. The method of claim 56 further comprises identifying immediate instructions within an outermost section in a program order of the immediate instructions within the outermost sections.

60. The method of claim 59 further comprises assigning data table locations for immediate data of the immediate instruction in the program order of the immediate instructions within the outermost sections.

61. The method of claim 60 further comprises storing immediate data of the immediate instructions in the assigned data table locations.

62. The method of claim 44 wherein identifying immediate data comprises:

grouping the program into sections, wherein a section comprises at least one instruction in a program path between branch nodes;

grouping the sections into outermost sections,

wherein in an outermost section comprises at least one section between corresponding start and end branch nodes that are unconditionally encountered during program execution; and

identifying immediate instructions within the outermost sections in a program order of outermost sections.

63. The method of claim 62 further comprises identifying the immediate instructions within an outermost section in a program order of the immediate instructions within the outermost sections.

64. The method of claim 63 further comprises assigning data table locations for immediate data of the immediate instruction in the program order of the immediate instructions within the outermost sections.

65. The method of claim 64 further comprises storing immediate data of the immediate instructions in the assigned data table locations.

66. The method of claim 62 further comprises identifying immediate instructions within an outermost

section in substantially a program order of the immediate instructions within the outermost sections, taking branching into account within the outermost sections.

67. The method of claim 66 further comprises assigning data table locations for immediate data of the immediate instruction in substantially the program order of the immediate instructions within the outermost sections.

68. The method of claim 67 further comprises storing immediate data of the immediate instructions in the assigned data table locations.

69. The method of claim 66 wherein identifying immediate instructions within an outermost section further comprises, if the outermost section comprises a conditional outermost section:

separating the outermost section into first and second paths;

identifying immediate instructions within the first path in substantially a program order of the immediate instructions within the first path; and

identifying immediate instructions within the

second path in substantially a program order of the immediate instructions within the first path.

70. The method of claim 69 further comprises assigning data table locations for immediate data of the immediate instruction in substantially the program order of the immediate instructions within the outermost sections.

71. The method of claim 70 further comprises storing immediate data of the immediate instructions in the assigned data table locations.

72. The method of claim 69 further comprises recursively analyzing the first and second paths to group the first and second paths into outermost sections to identify the immediate instructions in substantially the program order.

73. The method of claim 72 further comprises assigning data table locations for immediate data of the immediate instruction in substantially the program order of the immediate instructions within the outermost sections.

74. The method of claim 73 further comprises storing

immediate data of the immediate instructions in the assigned data table locations.

ABSTRACT

An efficient coding scheme is disclosed. The coding provides for the separation of immediate data from the instruction stream. A static flow analysis determines the immediate data in the program and assigns it a location for storage in a data table. The flow analysis also generates information directing the processor to the location of the immediate data in the data-order table. When an immediate instruction is encountered during execution, the processor retrieves the immediate data from the data-order table.

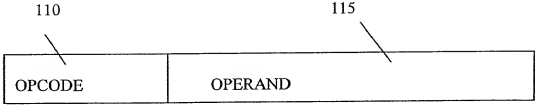


Fig. 1

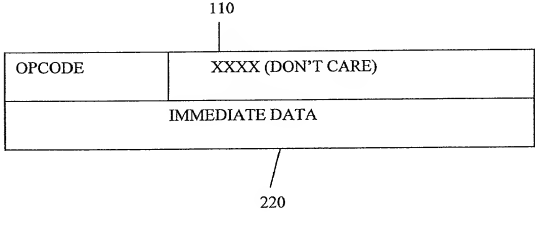


Fig. 2

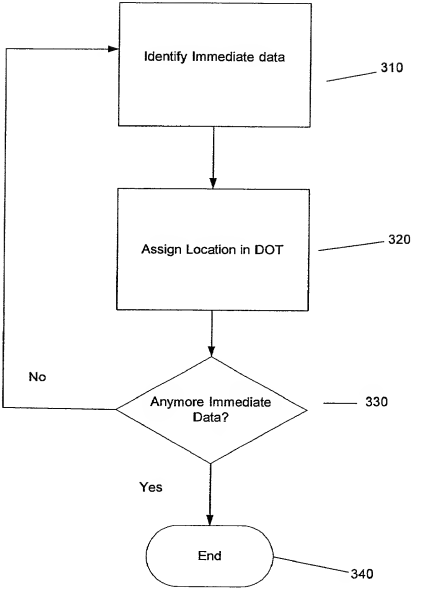


Fig. 3

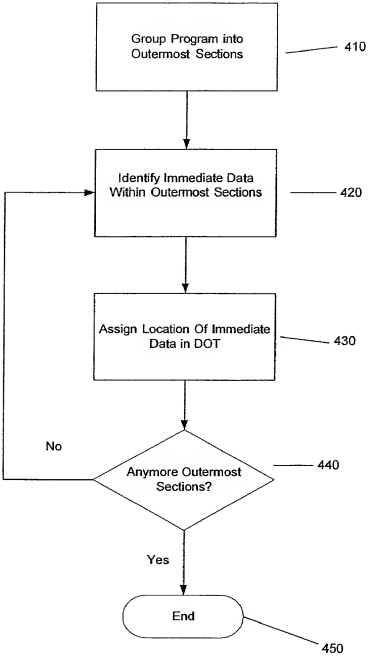

Group Program into Outermost Sections
410
Identify Immediate Data Within Outermost Sections
420
Assign Location Of Immediate Data in DOT
430
No
Anymore Outermost Sections?
440
Yes
End
450

Fig. 4

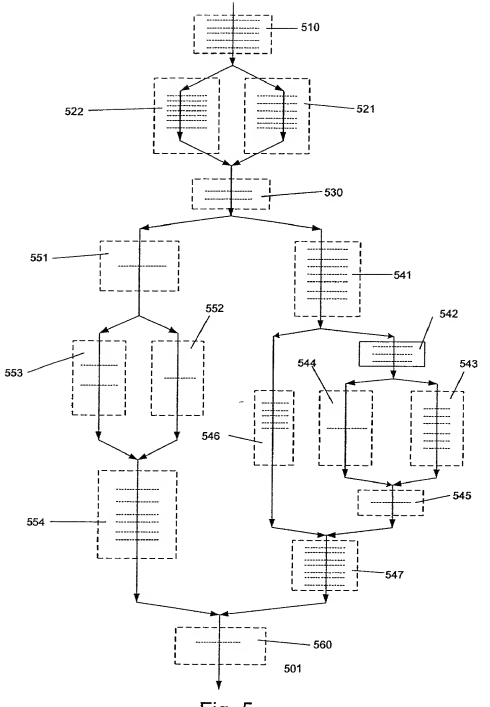


Fig. 5

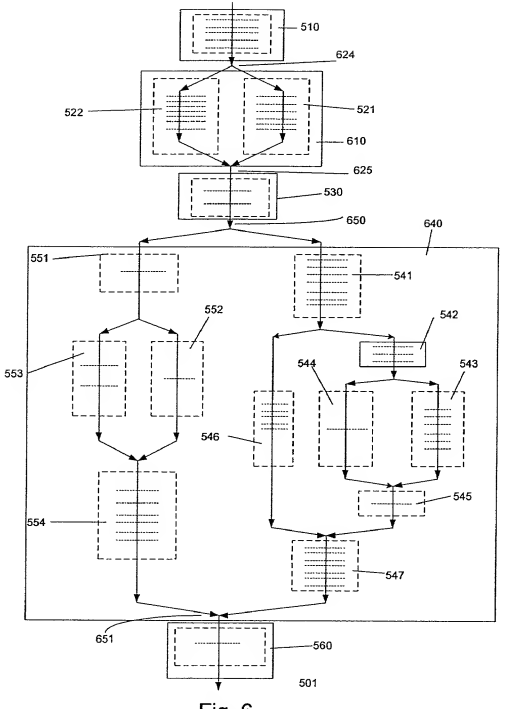
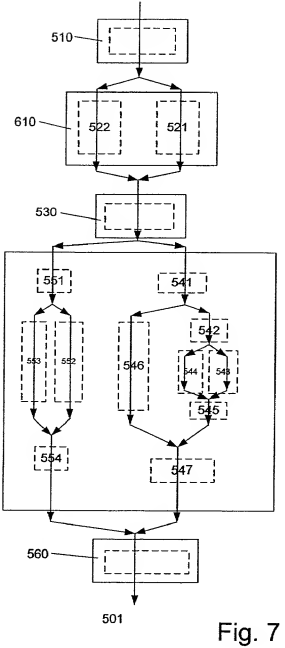

510
624
522
521
610
625
530
650
640
551
541
552
542
553
544
543
546
545
554
547
651
560
501

Fig. 6



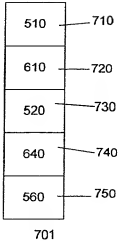


Fig. 7

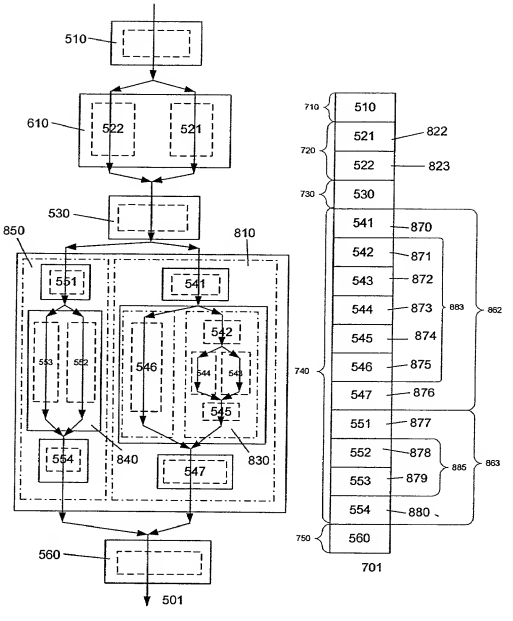


Fig. 8

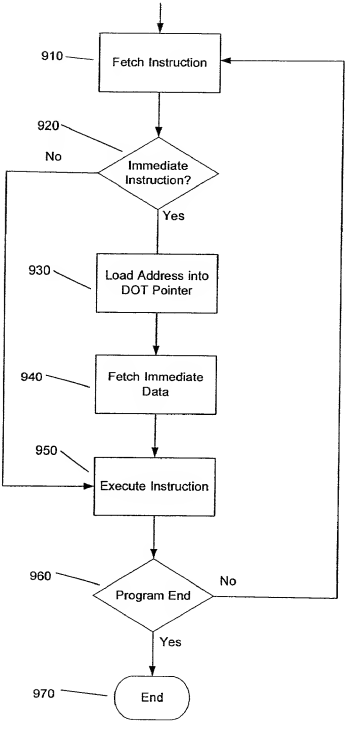


Fig. 9

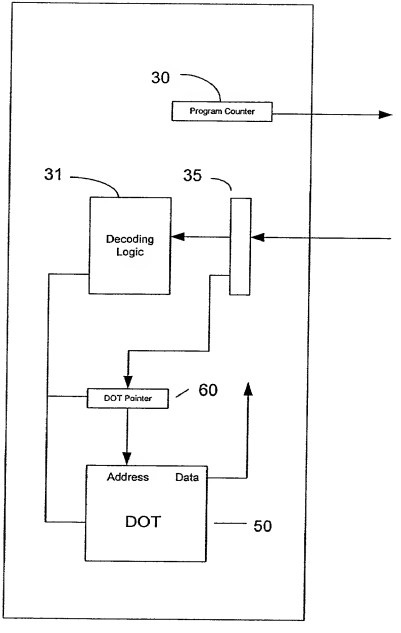


Fig. 10

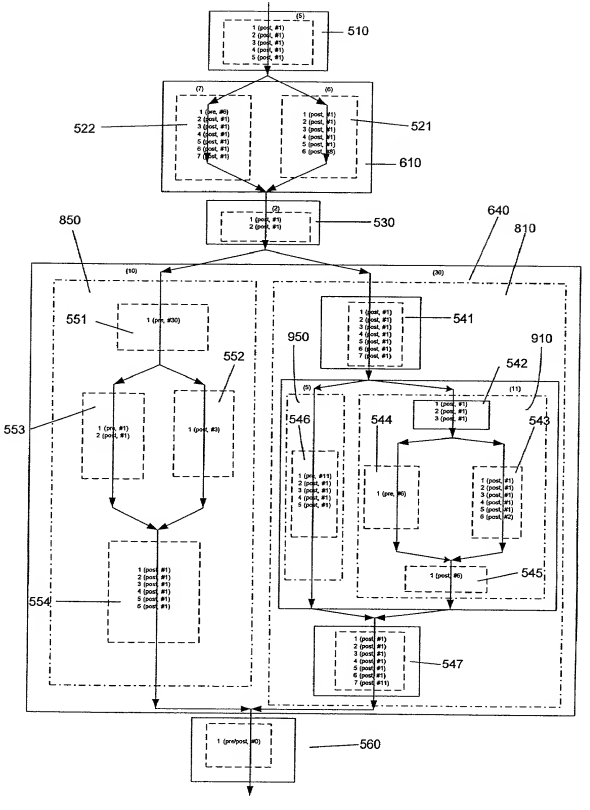


Fig. 11

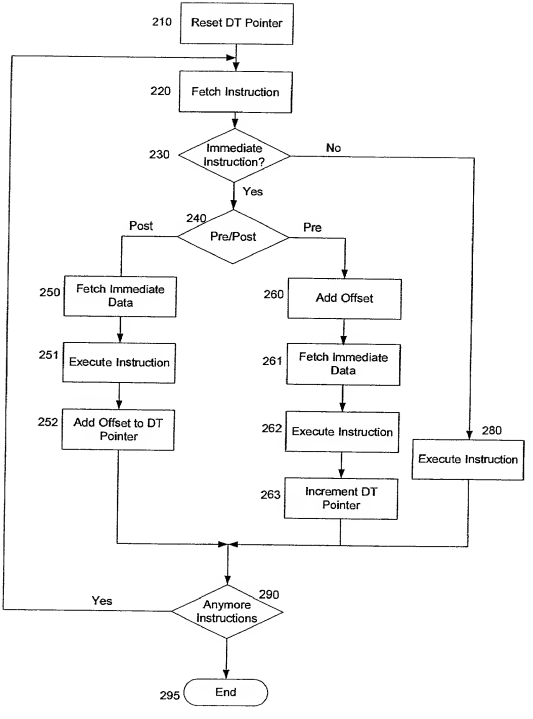


Fig. 12

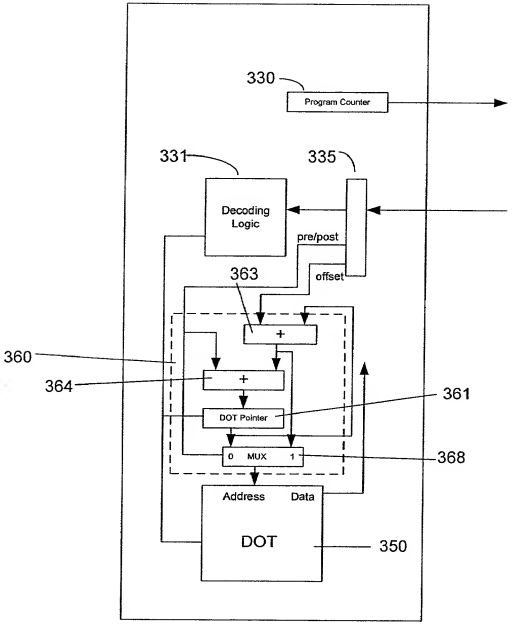


Fig. 13

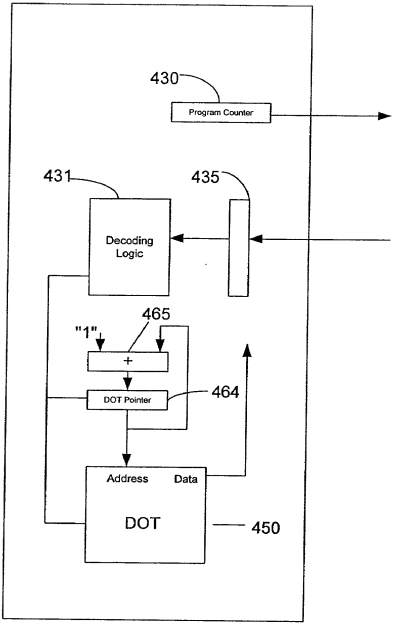


Fig. 14

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below, next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

EFFICIENT CODING IN PROCESSORS

the specification of which

____ is attached hereto.

__X__ was filed on November 14, 2001 as

Application Serial No. 09/979,572

and was amended on _____

(If applicable)

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above. I do not know and do not believe that the same was ever known or used in the United States of America before my invention thereof, or patented or described in any printed publication in any country before my invention thereof or more than one year prior to this application, that the same was not in public use or on sale in the United States of America more than one year prior to this application, and that the invention has not been patented or made the subject of an inventor's certificate issued before the date of this application in any country foreign to the United States of America on an application filed by me or my legal representatives or assigns more than twelve months prior to this application.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations, Section 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119, of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)			Priority Claimed	
PCT/SG99/00043	Singapore	17 May1999	X	____
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No
____	____	____	____	____
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56 which become available between the filing date of the prior application and the national or PCT international filing date of this application:

(Application Serial No.)	(Filing Date)	(Status -- patented, pending, abandoned)
(Application Serial No.)	(Filing Date)	(Status -- patented, pending, abandoned)

I hereby appoint Lawrence N. Ginsberg, Reg. No. 30,943, my patent attorney, with offices located at 907 Citrus Place, Newport Beach, California 92660-3227, telephone (949) 640-6261, with full power of substitution and revocation, to prosecute this application and to transact all business in the Patent and Trademark Office connected herewith.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

1-00

Full Name of Inventor **MANISH BHARDWAJ**

Inventor's Signature _____ Date 02/20/2001

Residence **Singapore, Singapore** SGX (City, State) Citizenship **India** (Country)

Address(residence) **#11-06 Aquamarine Tower, Bayshore Park**

Singapore

2-00

Full Name of Inventor **DEXTER CHIN**

Inventor's Signature _____ Date 12/05/2001

Residence **Singapore, Singapore** SGX (City, State) Citizenship **U.S.A** (Country)

Address(residence) **125 Meyer Road, #22-03 Makena**

Singapore 437936